20. Learning Probabilistic Models

This chapter discusses how intelligent agents can learn about the uncertain world by constructing probabilistic models from data. It emphasizes that learning can be viewed as a form of probabilistic inference, where the goal is to predict future events based on past observations and manage uncertainty using probabilistic reasoning.

20.1 Statistical Learning

Statistical learning involves using data (evidence) to infer hypotheses about the world, where hypotheses represent probabilistic theories explaining how the domain works. In this approach, the data serve as the observed values of certain random variables, while the hypotheses represent different probabilistic models.

- For instance, Bayesian learning methods use Bayes' rule to update the likelihood of different hypotheses as more data become available, allowing predictions that account for uncertainty and noise in the data.

Example: Consider a candy bag where the types of candies (cherry or lime) are unknown, and we want to predict the flavor of the next candy based on the flavors we have already observed. Bayesian learning calculates the probability of different hypotheses about the bag's candy composition and makes predictions by averaging over these hypotheses.

20.2 Learning with Complete Data

In this section, the focus is on learning from complete data, meaning every random variable in the data set has been observed. The goal is often to learn the parameters (numerical values) of a probabilistic model, such as the conditional probabilities in a Bayesian network or the parameters of other probabilistic models.

20.2.1 Maximum-likelihood Parameter Learning: Discrete Models

In maximum-likelihood learning, the task is to find the parameters that maximize the likelihood of the observed data. For discrete models, this often involves counting the occurrences of different outcomes and determining the parameter values that best explain the observed frequencies.

Example: In a model where candies can either be cherry or lime, the maximum-likelihood estimate of the proportion of cherry candies in a bag would simply be the observed ratio of cherry candies to the total number of candies.

20.2.2 Naive Bayes Models

Naive Bayes is a simplified Bayesian network model where the class variable (the label to predict) is the root node, and the attribute variables (features) are the leaves. The model assumes that, given the class. This assumption all attributes are conditionally independent of each other allows for efficient learning and inference, and despite being "naive," it often performs well in practice.

Example: In a spam email classifier, the class variable would be whether an email is spam or not, and the attributes might be the presence of specific words. The model assumes that given the spam status, the occurrence of each word is independent of the others.

20.2.3 Maximum-likelihood Parameter Learning: Continuous Models

In continuous models, such as those based on the Gaussian (normal) distribution, the maximum-likelihood learning task involves determining parameters like the mean (average) and standard deviation that best fit the observed data. The log-likelihood is used to simplify the maximization process since it turns products of probabilities into sums, making the math more tractable.

Example: Suppose you have a set of measurements of the heights of individuals. You can model the data with a Gaussian distribution and use maximum likelihood to estimate the average height (mean) and how spread out the heights are (standard deviation).

20.2.4 Bayesian Parameter Learning

Bayesian parameter learning extends maximum-likelihood learning by incorporating prior beliefs about the parameters before any data is observed. As new data is observed, these priors are updated to form a posterior distribution over the parameters. This approach helps avoid overfitting, particularly when data is sparse or noisy, by allowing prior beliefs to influence the model's predictions.

Example: If you have a prior belief that the bag of candy is likely to be mostly lime-flavored, Bayesian learning allows you to incorporate this belief and update it as you observe the actual flavors.

20.2.5 Learning Bayes Net Structures

Learning the structure of a Bayesian network (i.e., the relationships between variables) is a more complex task than parameter learning. This process often involves searching for the network structure that best fits the data while balancing complexity. Two key methods for learning structures are:

1. **Conditional Independence Tests**: Checking whether the relationships implied by the network structure are consistent with the observed data.
2. **Maximum Likelihood (or MAP) with Complexity Penalties**: Choosing the structure that maximizes the likelihood of the data while penalizing more complex models to avoid overfitting.

20.2.6 Density Estimation with Nonparametric Models

Nonparametric density estimation techniques do not assume a fixed parametric form for the probability distribution (like a Gaussian) but instead use methods such as nearest neighbors or kernel functions to estimate the underlying distribution directly from the data. These methods are useful when the form of the probability distribution is unknown.

Example: Given a set of two-dimensional points, k-nearest neighbors might estimate the density around a point by measuring how many neighbors fall within a given distance. Kernel density estimation would use smooth functions, like Gaussians, centered at each data point to estimate the overall distribution.

20.3 Learning with Hidden Variables: The EM Algorithm

The Expectation-Maximization (EM) algorithm is a powerful technique used to learn probabilistic models when some variables are hidden (unobserved). The EM algorithm iterates between two steps:

- **E-step (Expectation)**: Estimates the distribution of the hidden variables based on the observed data and current model parameters.
- **M-step (Maximization)**: Updates the model parameters by maximizing the expected log-likelihood calculated in the E-step.

20.3.1 Unsupervised Clustering: Learning Mixtures of Gaussians

Unsupervised clustering involves identifying different categories (or clusters) in data without knowing the labels in advance. The EM algorithm is often used to fit a mixture of Gaussians model, where the data are assumed to come from a mixture of multiple Gaussian distributions. Each data point is assigned probabilistically to one of these distributions, and the parameters of each distribution are updated iteratively.

Example: If you have a dataset of star observations, EM could be used to cluster the stars into groups (like red giants and white dwarfs) by fitting a mixture of Gaussian distributions to the data.

20.3.2 Learning Bayesian Networks with Hidden Variables

The EM algorithm can also be used to learn Bayesian networks where some variables are hidden. These hidden variables often represent underlying causes or latent factors that explain the observed data. In the E-step, the hidden variables are inferred, and in the M-step, the network parameters are updated.

Example: In a medical diagnosis problem, diseases may be hidden variables, while the observed variables are symptoms. The EM algorithm can infer the most likely diseases and adjust the model to best fit the observed symptoms.

20.3.3 Learning Hidden Markov Models

Hidden Markov Models (HMMs) represent processes where the system transitions between different hidden states over time. EM, specifically the Baum-Welch algorithm, is used to learn the transition probabilities and the emission probabilities (how states generate observations) from observed sequences of data.

Example: In speech recognition, the hidden states could represent phonemes (basic sound units), and the observed data are the sound signals. The EM algorithm can learn how the phonemes transition over time and how they correspond to different sound patterns.

20.3.4 The General Form of the EM Algorithm

The general form of the EM algorithm is applicable to a wide variety of models beyond mixtures of Gaussians or hidden Markov models. The E-step computes the expected values of hidden variables, and the M-step maximizes the likelihood of the data given these expectations. This general framework can be customized for many different types of probabilistic models.

20.3.5 Learning Bayes Net Structures with Hidden Variables

Learning both the structure and parameters of a Bayesian network with hidden variables is a challenging task. The structure search may involve adding or removing variables and connections while optimizing the likelihood of the data, often requiring complex search algorithms or sampling techniques (like MCMC). This process allows the discovery of latent variables that can simplify the model and improve its fit to the data.

Algorithms

1. **Bayesian Learning Algorithm**

Bayesian learning is based on updating the probability of a hypothesis as more data becomes available, using **Bayes' Theorem.** Instead of relying on a single hypothesis, it evaluates all possible hypotheses, weighted by their probability, and makes predictions based on these.

- **Bayes' Rule:** Given a hypothesis h_i and data d, the probability of the hypothesis given the data is computed as:

  $$P(h_i|d) = \alpha P(d|h_i)P(h_i)$$

  Where:

  - $P(h_i|d)$ is the posterior probability of the hypothesis.
  - $P(d|h_i)$ is the likelihood of the data given the hypothesis.
  - $P(h_i)$ is the prior probability of the hypothesis.
  - α is a normalizing constant.

  The prediction for an unknown value X is computed by averaging over all hypotheses:

  $$P(X|d) = \sum_i P(X|h_i)P(h_i|d)$$

- **Maximum A Posteriori (MAP) Hypothesis:** In practice, instead of averaging over all hypotheses, sometimes we choose the hypothesis with the highest posterior probability (MAP hypothesis). This simplifies the prediction by selecting the most probable explanation for the data.

2. **Maximum-Likelihood Learning Algorithm**

Maximum-Likelihood Estimation (MLE) is an algorithm used to find the parameters of a probabilistic model that maximize the likelihood of the observed data.

- **Algorithm:**
  1. **Write the likelihood** of the observed data D as a function of the parameters θ: $P(D|\theta)$.
  2. **Take the log of the likelihood** function to simplify computation: $\log P(D|\theta)$.
  3. **Differentiate** the log-likelihood with respect to the parameters.
  4. **Set the derivatives to zero** to find the parameter values that maximize the log-likelihood.

Example (Discrete case): If you have a coin-flipping model, and you observe the outcomes of a series of flips, MLE would estimate the probability of heads (parameter θ) by counting the fraction of heads in the observed data.

3. **Expectation-Maximization (EM) Algorithm**

The **EM algorithm** is used to learn probabilistic models with hidden (unobserved) variables. It is an iterative method that alternates between two steps:

- **E-step (Expectation):** Compute the expected values of the hidden variables given the observed data and the current parameter estimates.
- **M-step (Maximization):** Update the parameters to maximize the expected log-likelihood computed in the E-step.

The EM algorithm is widely used in problems like **Gaussian Mixture Models (GMMs)**, **Hidden Markov Models (HMMs)**, and **Bayesian networks.**

- **Algorithm:**
  1. **Initialize** the parameters randomly or based on prior knowledge.
  2. **E-step:** For each data point, compute the probability that the data point was generated by each component of the model (e.g., each Gaussian in a mixture model).
  3. **M-step:** Use the probabilities from the E-step to update the model's parameters (e.g., means and variances of the Gaussians).
  4. **Repeat** steps 2 and 3 until convergence (when the parameters stop changing significantly).

Example (Mixture of Gaussians):

- In a Gaussian Mixture Model (GMM), the data is assumed to come from a mixture of several Gaussian distributions. The EM algorithm estimates the means, variances, and mixture weights of these Gaussians by iteratively computing which Gaussian most likely generated each data point (E-step) and updating the parameters of the Gaussians (M-step).

4. Naive Bayes Algorithm

The **Naive Bayes classifier** is a simple probabilistic classifier based on Bayes' theorem. It assumes that the features (attributes) are conditionally independent given the class label. This assumption simplifies the computation of the likelihood.

- **Algorithm:**
  1. **Calculate prior probabilities** for each class.
  2. **Calculate conditional probabilities** for each feature given the class.
  3. **Classify** a new instance by selecting the class that maximizes the posterior probability, computed as:

$$P(C|x_1, x_2, \ldots, x_n) \propto P(C) \prod_{i=1}^{n} P(x_i|C)$$

  Where $x_1, x_2, \ldots, x_n$ are the features of the instance, and $P(C|x_1, x_2, \ldots, x_n)$ is the posterior probability of class C.

- **Example:** In spam email classification, the algorithm computes the probability that an email is spam based on the presence of certain words, assuming each word independently contributes to the probability.

5. Learning Bayes Net Structures

Bayesian networks represent the conditional dependencies between random variables in a graphical structure. Learning the structure of a Bayes net involves searching for the best network structure that explains the data.

- **Search Algorithm:**
  1. **Start** with an empty or initial network structure.
  2. **Iteratively add, remove, or reverse edges** between nodes in the network.
  3. **Score each structure** based on the likelihood of the data given the network structure, often penalizing for complexity (to avoid overfitting).
  4. **Choose the structure** with the best score.

There are two common scoring methods:

- **Maximum-likelihood scoring** (without hidden variables).
- **Bayesian scoring** (with hidden variables), which includes priors over structures and parameters.

Algorithm examples: Hill-climbing or greedy search, Simulated Annealing, and Markov Chain Monte Carlo (MCMC) are commonly used for structure learning in Bayesian networks.

6. **k-Nearest Neighbors (k-NN) for Density Estimation**

In **nonparametric density estimation**, k-NN is used to estimate the density of data points in continuous spaces. The idea is to estimate the probability density function by counting the number of neighboring data points within a certain distance from a query point.

- **Algorithm:**
  1. For a given query point, find its **k nearest neighbors** from the dataset.
  2. Estimate the density as the inverse of the volume of the region that contains these k neighbors.

Example: In a 2D space, the algorithm would estimate the density of a region by calculating how tightly clustered data points are around a query point.

7. **Hidden Markov Model (HMM) Learning**

An **HMM** is a probabilistic model used to represent systems that transition between hidden states over time. The Baum-Welch algorithm, a specific form of the EM algorithm, is commonly used to learn the parameters of HMMs from observed sequences.

- **Algorithm:**
  1. **E-step:** Use the **forward-backward algorithm** to compute the expected counts of transitions between hidden states based on the observed data.
  2. **M-step:** Update the transition and emission probabilities using the expected counts from the E-step.

Example: In speech recognition, HMMs are used to model sequences of sounds, where the hidden states correspond to phonemes, and the observations are the acoustic signals.

8. **Kernel Density Estimation (KDE)**

KDE is another nonparametric method used for estimating the probability density function of a random variable. Instead of assuming a specific parametric distribution, KDE estimates the density by placing a smooth kernel (often Gaussian) over each data point.

- **Algorithm:**
  1. Place a kernel function over each data point.
  2. The estimated density at a point is the sum of the contributions from all kernels.

$$P(x) = \frac{1}{N}\sum_{i=1}^{N} K(x - x_i)$$

Where K is a kernel function (such as Gaussian), and N is the number of data points.

Example: KDE can be used to estimate the distribution of heights in a population, where the data points are individual height measurements, and the kernel function smooths out the distribution.